PROJECT#2
EVALUATING THE IMPACT OF VARYING BRANCH PREDICTION PARAMETERS IN AN X86 ARCHITECTURE-BASED MICROPROCESSOR USING GEM5 SIMULATOR

UNDER: PROF DR. KANAD BASU

TEAM: (100/100)
ABHINAY DWADASI (AXD220054)

YAGNA SRINIVASA HARSHA ANNADATA (YXA210024)

(CE6304) CLASS OF 2023-24 (SPRING SEMESTER)

UT DALLAS
The University of Texas at Dallas

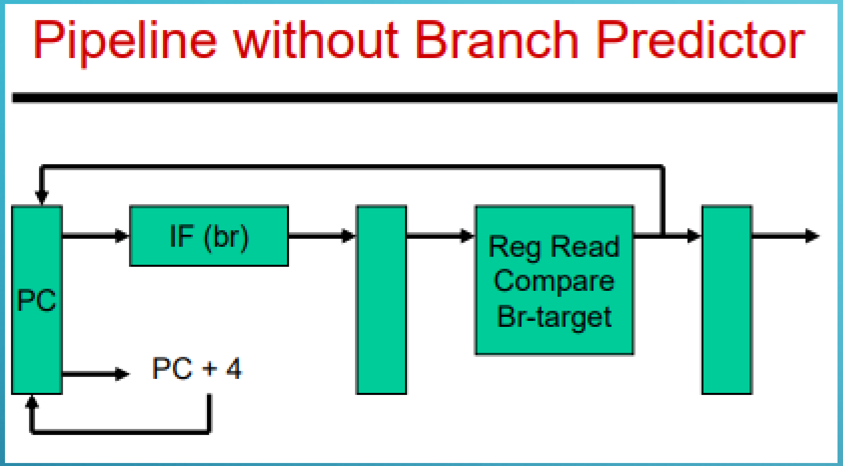
PROJECT DESCRIPTION:

- Evaluating the efficiency of various branch predictors on X86 MP using GEM5.
- **CPU type**: Timing Simple CPU.
- **CPU configuration:**
- 128 kB for L1 Instruction cache.
- 128 kB for L1 Data cache.
- 1024 kB for L2 cache.
- **Associativity:**
- For L1D and L1I: 2-way Set associative.
- For L2: 4-way set associative.
- Cache Block Size: 64 bytes.
- **Benchmarks:**
- 401.bzip2, 429.mcf, 456.hmmr. 458.sjeng, 470.lbm
- **Branch Predictors:**
- 2 bit-Local branch predictor.
- Bi_mode branch predictor
- Tournament predictor.
- Standard Instruction Load: 500000000

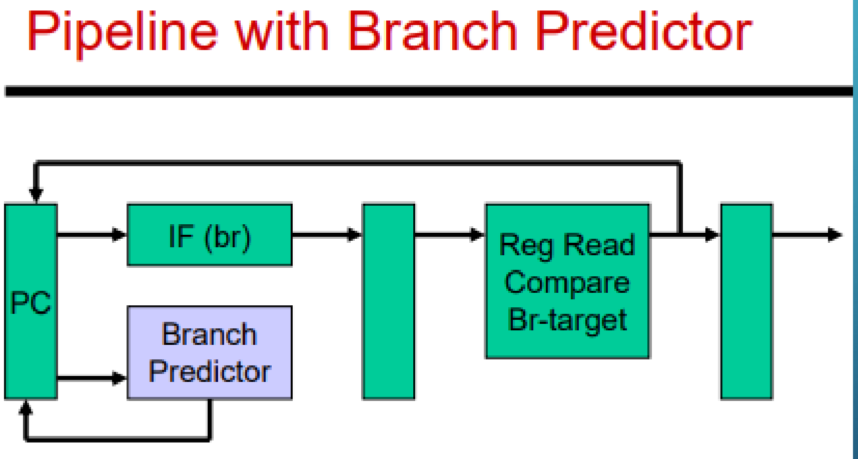
WHAT IS BRANCH PREDICTION ?

- Branch prediction is a technique used in computer architecture to improve the performance of pipelined processors.
- Branch prediction is used to guess the target address of the branch instruction before it is determined. This is done with the help of a branch predictor.
- A branch predictor makes an educated guess based on the past behavior of the program or code pattern.

HOW CAN A BRANCH PREDICTOR AFFECT THE PERFORMANCE OF A PROCESSOR?



Pipeline without Branch Predictor



Pipeline with Branch Predictor

- The goal of branch prediction is to reduce the number of pipeline stalls caused by branch instructions and improve the overall performance of the processor. As you can see above, instead of fetching fall through instructions, a branch predictor will automatically jump to the address based on the algorithm used by the implemented predictor that reduces stalls for flushing and re-fetching appropriate instructions.

** cited images from University of UTAH CS6810 slides.

TYPES OF BRANCH PREDICTORS

Bi_Mode Branch Predictor	**Local Branch Predictor**	**Tournament Branch Predictor**
• The bi_mode predictor is a type of branch predictor that maintains a two tables of branch buffers; taken and not taken branch outcomes. • Second level two-bit counter table is divided into two along with the branch address and the branch predictor state machine(PC). • Branch address is used to make the final call • This predictor works well for programs with equal probability of branch taken and not taken.	• The local predictor is a more sophisticated type of branch predictor that considers the behavior of the branch instructions in a local region of the program. • It maintains a table of branch histories for each region of the program and uses that information to predict the outcome of a branch instruction within that region. • More effective for programs with more complex control flow structures.	• The tournament predictor is a hybrid type of branch predictor that combines the strengths of the bimodal and local predictors. • It maintains both a bimodal table and a local history table and selects the predictor that has been more accurate for the given branch instruction in the past. • The tournament predictor uses a meta-predictor to choose the most accurate predictor for a particular branch instruction.

GEM5 CONFIGURATION

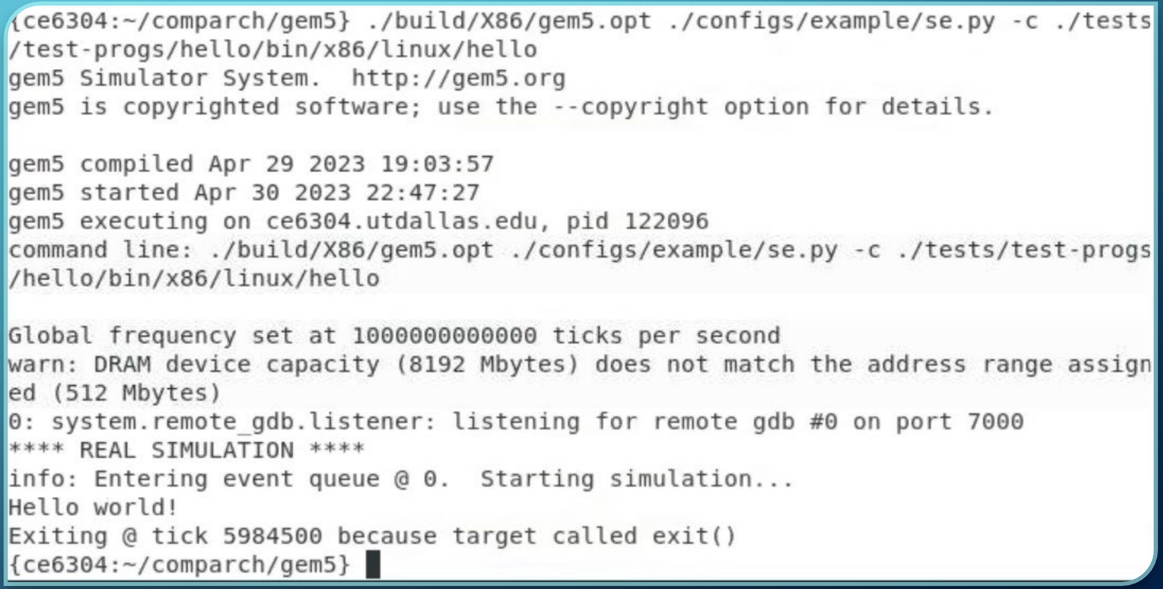
- Gem5 was accessed from the virtual server CE6304.utdallas.edu
- Files have been copied from Gem5 directory (/usr/local/gem5). Configuration files have been moved to local directory using “cp rf-” command.
- We used “scons build/X86/gem5.opt” after copying the files to build and compile Gem5.
- Similar to Project 1, we still run the built CPU in system emulation mode using the following Python script: “/configs/example/se.py”.
- **Minor challenge that we have faced:**
  While initially setting up gem5, we used the same files project 1, we were not able to build gem5 using previous files from some reason and hence, all the old files were deleted, and a fresh copy of the dependency files were copied. Post which we were able to successfully build and compile Gem5 simulator.

CHALLENGES FACED

- The initial setup takes a longer time to build approximately 2 hours.
- After every change to the Branch Predictor type or value we had to build and then run the run.sh file.
- Hence every execution has taken approximately 30 min to complete except for 470.lbm benchmark.
- for adding BTB Miss percent and Branch Misprediction percentage to stats.txt, there was difficulty initially to find the files which are to be changed. We later used **grep -r “numBranches” --include="*.cc" --include="*.hh"** to find the files and **grep -r “BTBHitPct” --include="*.cc" --include="*.hh"**

INITIAL SETUP

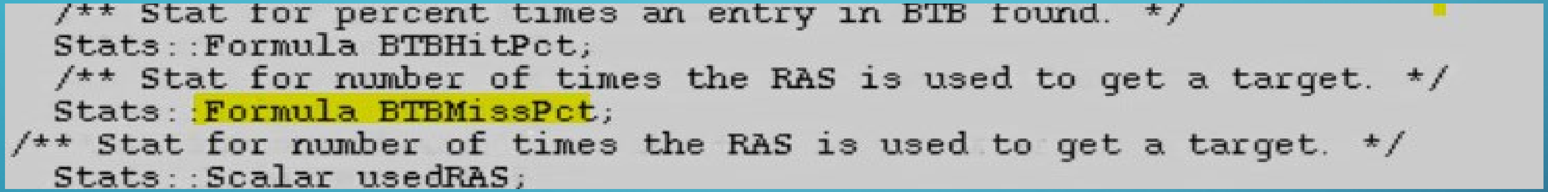
- We then downloaded the required benchmarks from github using: https://github.com/timberjack/Project1_SPEC.
- Made changes to the source code according to the requirement.
- Then we used **scons build/X86/gem5.opt** for building the executable file.
- Ran the example programs for verification as shown in the figure.



```
{ce6304:~/comparch/gem5} ./build/X86/gem5.opt ./configs/example/se.py -c ./tests
/test-progs/hello/bin/x86/linux/hello
gem5 Simulator System.  http://gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Apr 29 2023 19:03:57
gem5 started Apr 30 2023 22:47:27
gem5 executing on ce6304.utdallas.edu, pid 122096
command line: ./build/X86/gem5.opt ./configs/example/se.py -c ./tests/test-progs
/hello/bin/x86/linux/hello

Global frequency set at 1000000000000 ticks per second
warn: DRAM device capacity (8192 Mbytes) does not match the address range assign
ed (512 Mbytes)
0: system.remote_gdb.listener: listening for remote gdb #0 on port 7000
**** REAL SIMULATION ****
info: Entering event queue @ 0.  Starting simulation...
Hello world!
Exiting @ tick 5984500 because target called exit()
{ce6304:~/comparch/gem5} 
```

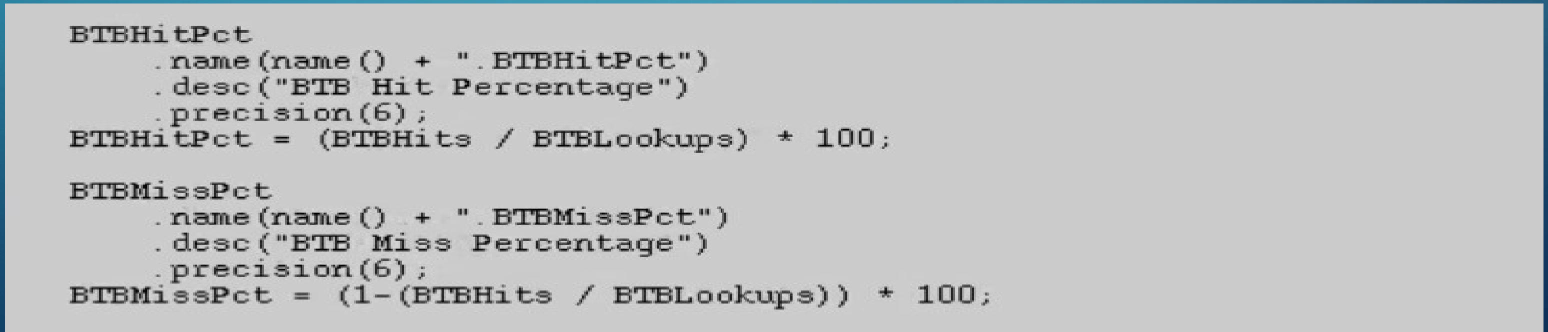
ADDING FORMULA FOR REFLECTING BTB MISS PERCENT IN STATS.TXT FILE

- Following are the changes that have been made to the files in the following paths:
- File path 1: ~gem5/src/cpu/pred/bpred_unit.hh



```
  /** Stat for percent times an entry in BTB found. */
  Stats::Formula BTBHitPct;
  /** Stat for number of times the RAS is used to get a target. */
  Stats::Formula BTBMissPct;
/** Stat for number of times the RAS is used to get a target. */
  Stats::Scalar usedRAS;
```

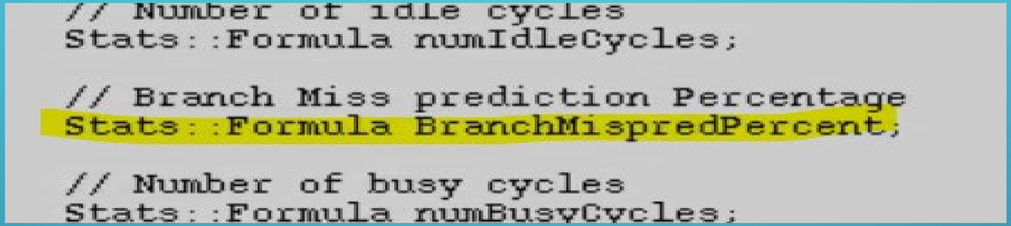
- File Path2: ~gem5/src/cpu/pred/bpred_unit.cc



```
    BTBHitPct
        .name(name() + ".BTBHitPct")
        .desc("BTB Hit Percentage")
        .precision(6);
    BTBHitPct = (BTBHits / BTBLookups) * 100;

    BTBMissPct
        .name(name() + ".BTBMissPct")
        .desc("BTB Miss Percentage")
        .precision(6);
    BTBMissPct = (1-(BTBHits / BTBLookups)) * 100;
```

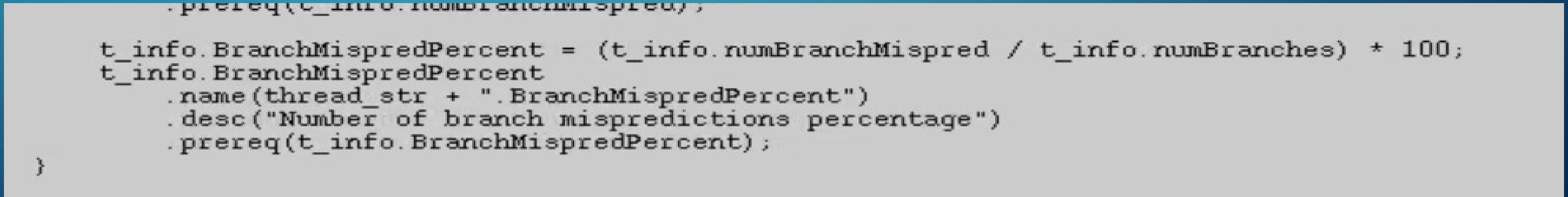
ADDING FORMULA FOR REFLECTING BRANCH MISPREDICTION IN STATS.TXT FILE

- Following are the changes that have been made to the files in the following paths:
- File path: ~/gem5/src/cpu/simple/exec_context.hh



```
    // Number of idle cycles
    Stats::Formula numIdleCycles;

    // Branch Miss prediction Percentage
    Stats::Formula BranchMispredPercent;

    // Number of busy cycles
    Stats::Formula numBusyCycles;
```

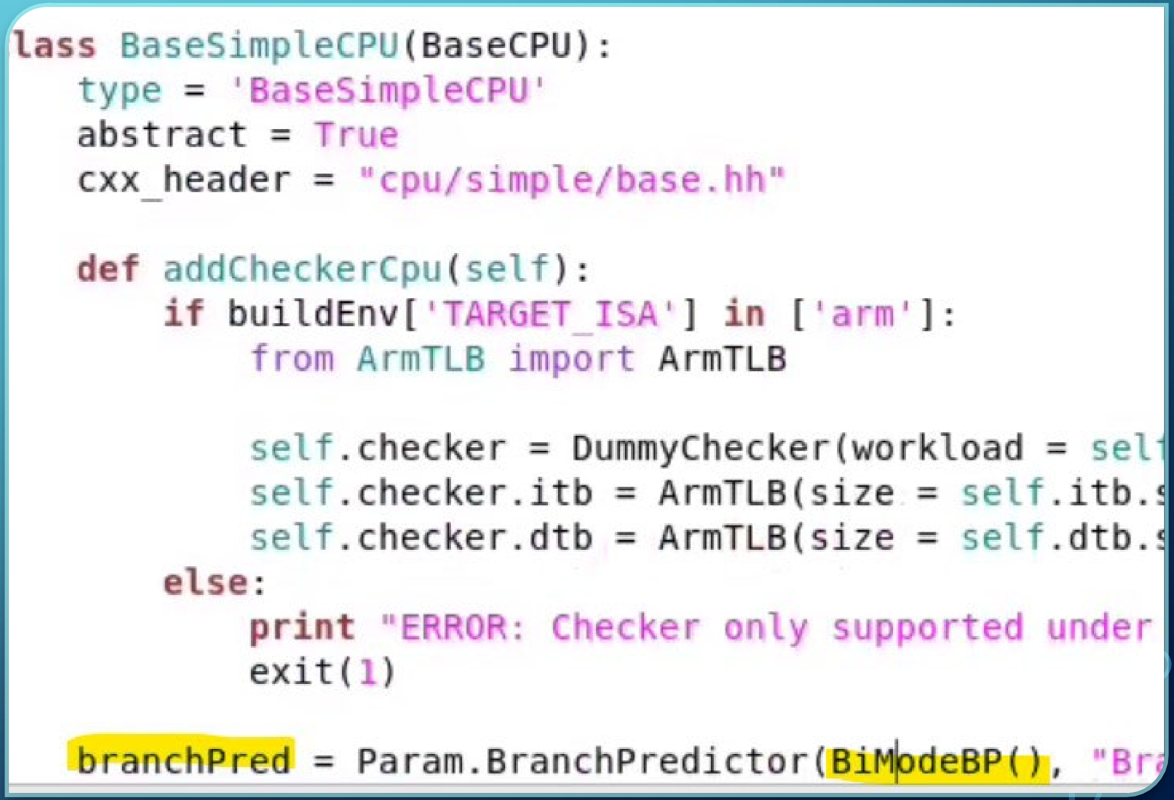
- File path: ~/gem5/src/cpu/simple/base.cc



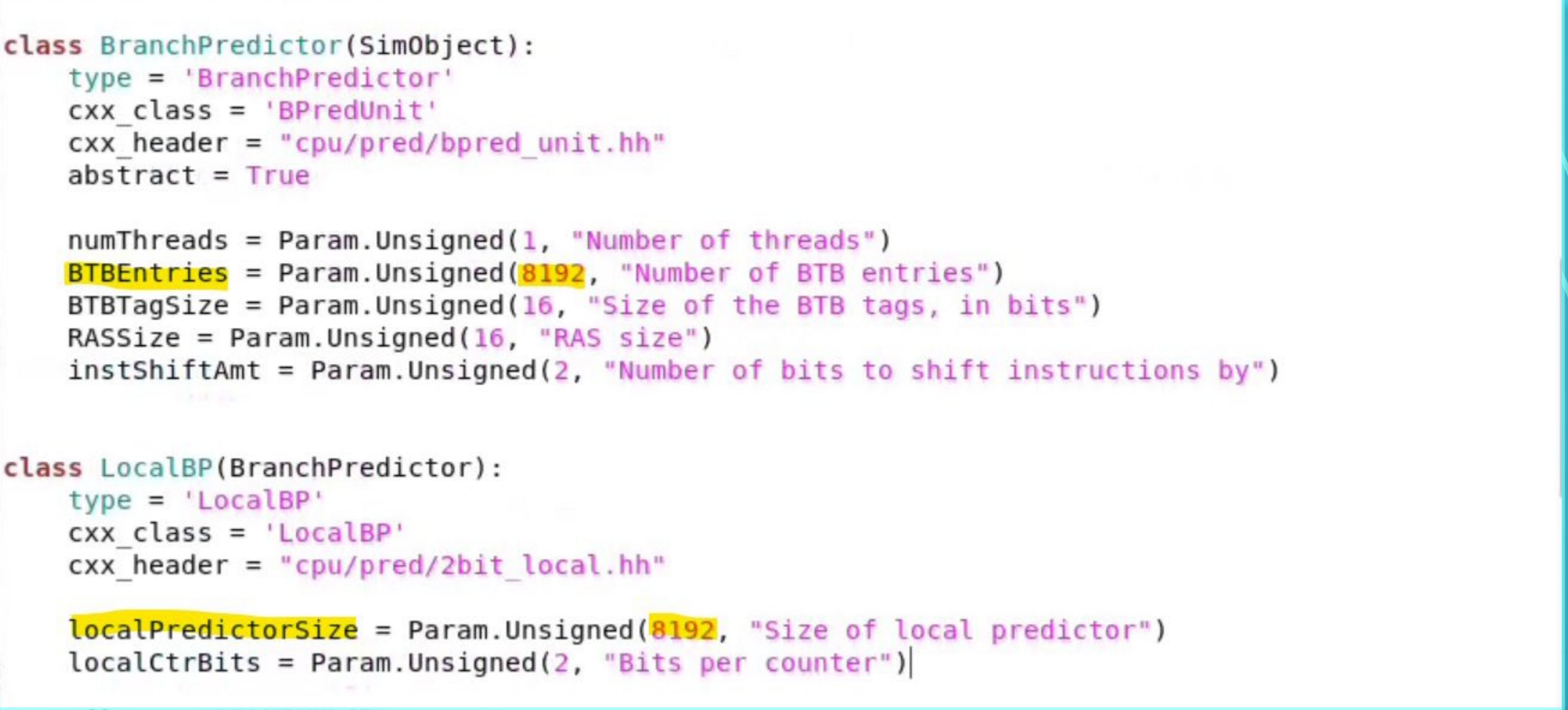
```
            .prereq(t_info.numBranchMispred);

        t_info.BranchMispredPercent = (t_info.numBranchMispred / t_info.numBranches) * 100;
        t_info.BranchMispredPercent
            .name(thread_str + ".BranchMispredPercent")
            .desc("Number of branch mispredictions percentage")
            .prereq(t_info.BranchMispredPercent);
  }
```

CHANGES FOR THE NAME OF BRANCH PREDICTOR

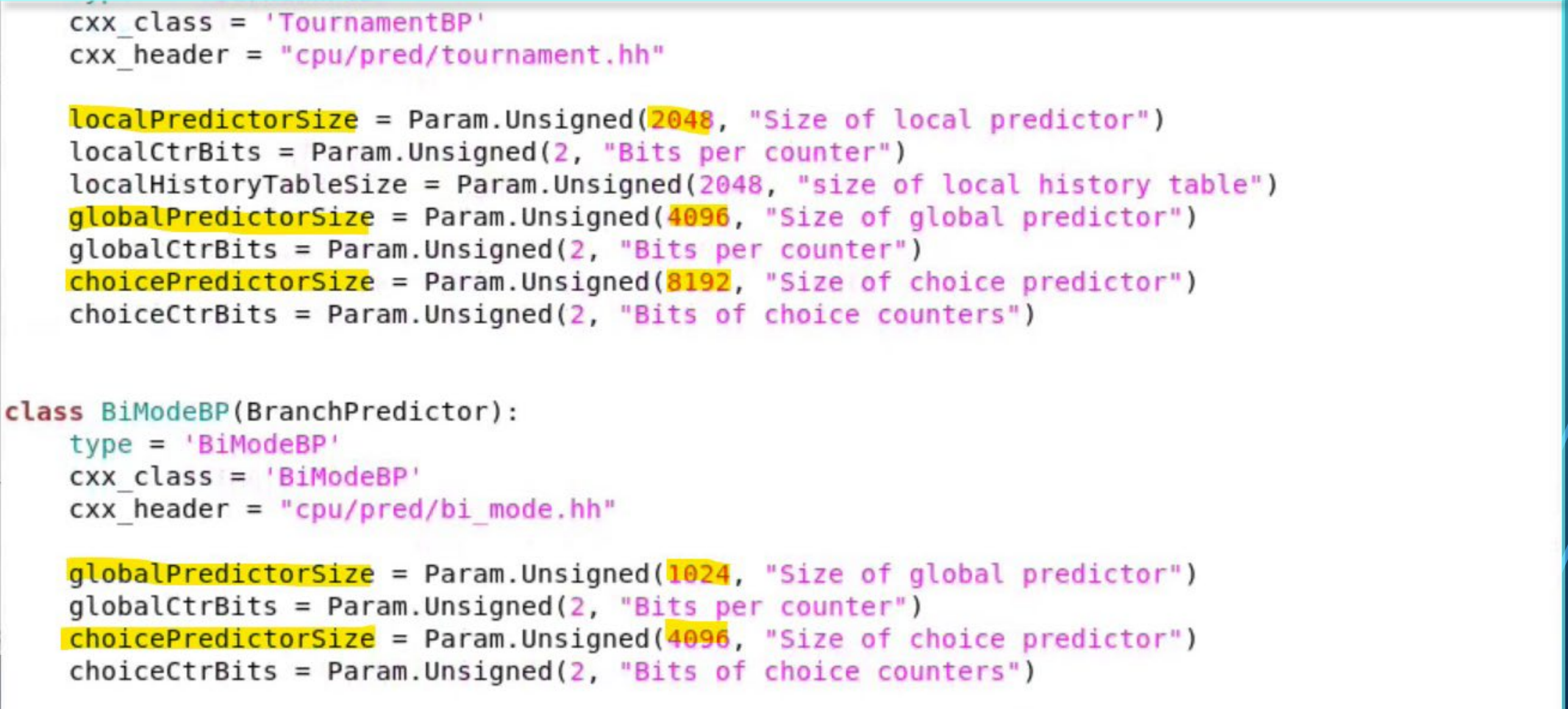
- FilePath: ~/gem5/src/cpu/simple/**BaseSimpleCPU.py**
- We give the Branch Predictor required in the highlighted space.



```
lass BaseSimpleCPU(BaseCPU):
    type = 'BaseSimpleCPU'
    abstract = True
    cxx_header = "cpu/simple/base.hh"

    def addCheckerCpu(self):
        if buildEnv['TARGET_ISA'] in ['arm']:
            from ArmTLB import ArmTLB

            self.checker = DummyChecker(workload = sel
            self.checker.itb = ArmTLB(size = self.itb.
            self.checker.dtb = ArmTLB(size = self.dtb.
        else:
            print "ERROR: Checker only supported under
            exit(1)

    branchPred = Param.BranchPredictor(BiModeBP(), "Br
```



```
class BranchPredictor(SimObject):
    type = 'BranchPredictor'
    cxx_class = 'BPredUnit'
    cxx_header = "cpu/pred/bpred_unit.hh"
    abstract = True

    numThreads = Param.Unsigned(1, "Number of threads")
    BTBEntries = Param.Unsigned(8192, "Number of BTB entries")
    BTBTagSize = Param.Unsigned(16, "Size of the BTB tags, in bits")
    RASSize = Param.Unsigned(16, "RAS size")
    instShiftAmt = Param.Unsigned(2, "Number of bits to shift instructions by")


class LocalBP(BranchPredictor):
    type = 'LocalBP'
    cxx_class = 'LocalBP'
    cxx_header = "cpu/pred/2bit_local.hh"

    localPredictorSize = Param.Unsigned(8192, "Size of local predictor")
    localCtrBits = Param.Unsigned(2, "Bits per counter")
```



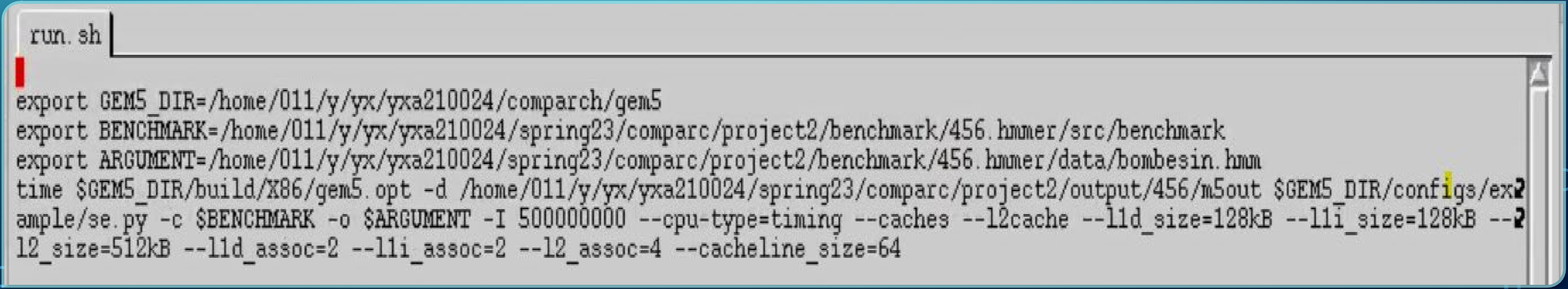
```
    cxx_class = 'TournamentBP'
    cxx_header = "cpu/pred/tournament.hh"

    localPredictorSize = Param.Unsigned(2048, "Size of local predictor")
    localCtrBits = Param.Unsigned(2, "Bits per counter")
    localHistoryTableSize = Param.Unsigned(2048, "size of local history table")
    globalPredictorSize = Param.Unsigned(4096, "Size of global predictor")
    globalCtrBits = Param.Unsigned(2, "Bits per counter")
    choicePredictorSize = Param.Unsigned(8192, "Size of choice predictor")
    choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")


class BiModeBP(BranchPredictor):
    type = 'BiModeBP'
    cxx_class = 'BiModeBP'
    cxx_header = "cpu/pred/bi_mode.hh"

    globalPredictorSize = Param.Unsigned(1024, "Size of global predictor")
    globalCtrBits = Param.Unsigned(2, "Bits per counter")
    choicePredictorSize = Param.Unsigned(4096, "Size of choice predictor")
    choiceCtrBits = Param.Unsigned(2, "Bits of choice counters")
```

CHANGES FOR BRANCH PREDICTOR VALUES

- File Path for the values of Branch Predictor ~/gem5/src/cpu/pred/**BranchPredictor.py**
- Values of the Branch Predictors are given according to the requirement in the highlighted space

RUNNING GEM5 SCRIPT

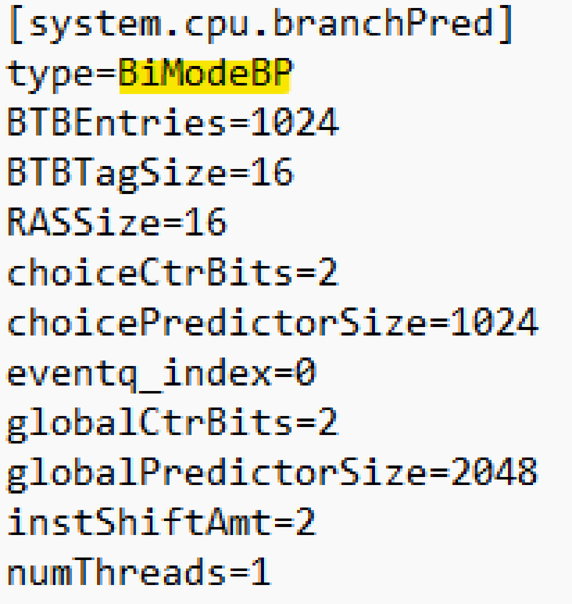
- We have used a run.sh script to run the gem5 command in each benchmark.
- Filepath: ~/benchmark/456.hmmer/**run.sh**
- We modified the run.sh file according to the requirement as shown in figure.
- We have allowed permission to run executable files by using **chmod +x run.sh**
- On terminal we have run the executable file using **./run.sh**



```
export GEM5_DIR=/home/011/y/yx/yxa210024/comparch/gem5
export BENCHMARK=/home/011/y/yx/yxa210024/spring23/comparc/project2/benchmark/456.hmmer/src/benchmark
export ARGUMENT=/home/011/y/yx/yxa210024/spring23/comparc/project2/benchmark/456.hmmer/data/bombesin.hmm
time $GEM5_DIR/build/X86/gem5.opt -d /home/011/y/yx/yxa210024/spring23/comparc/project2/output/456/m5out $GEM5_DIR/configs/ex
ample/se.py -c $BENCHMARK -o $ARGUMENT -I 500000000 --cpu-type=timing --caches --l2cache --l1d_size=128kB --l1i_size=128kB --
l2_size=512kB --l1d_assoc=2 --l1i_assoc=2 --l2_assoc=4 --cacheline_size=64
```

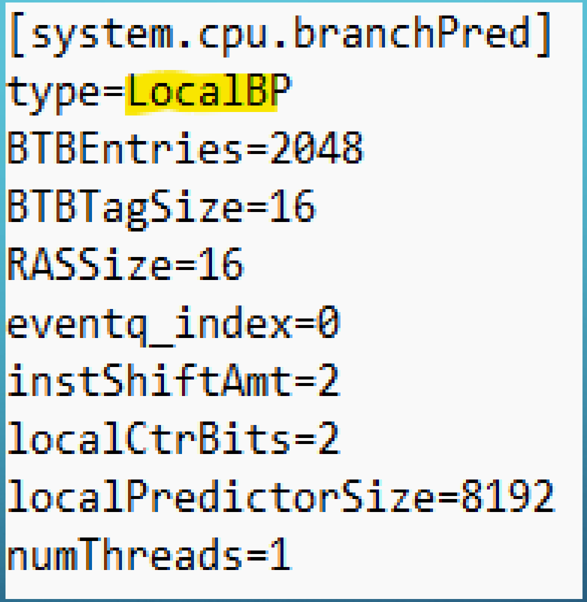
CHANGES OBSERVED IN CONFIG.INI FILE AFTER ADDING BRANCH PREDICTOR SUPPORT AND MAKING CHANGES FOR THE PARAMETERS

BI_MODE PREDICTOR



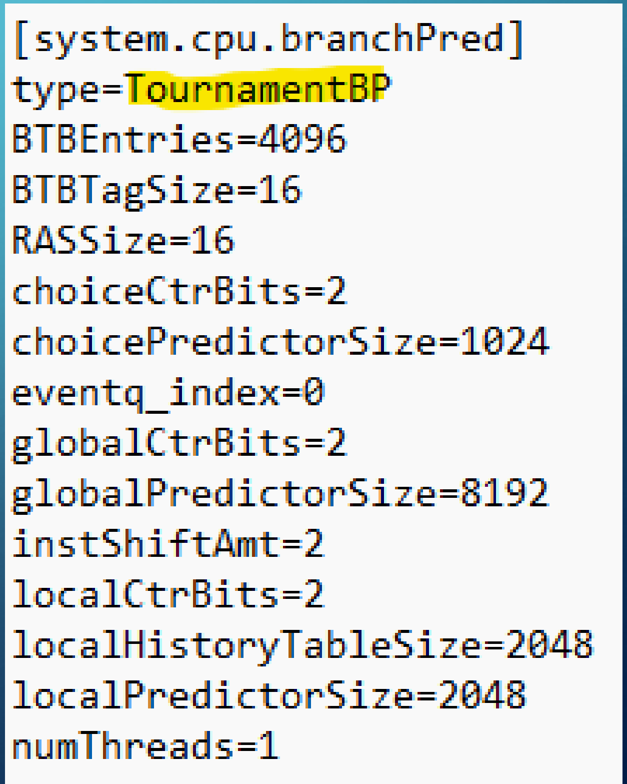
```
[system.cpu.branchPred]
type=BiModeBP
BTBEntries=1024
BTBTagSize=16
RASSize=16
choiceCtrBits=2
choicePredictorSize=1024
eventq_index=0
globalCtrBits=2
globalPredictorSize=2048
instShiftAmt=2
numThreads=1
```

LOCAL PREDICTOR



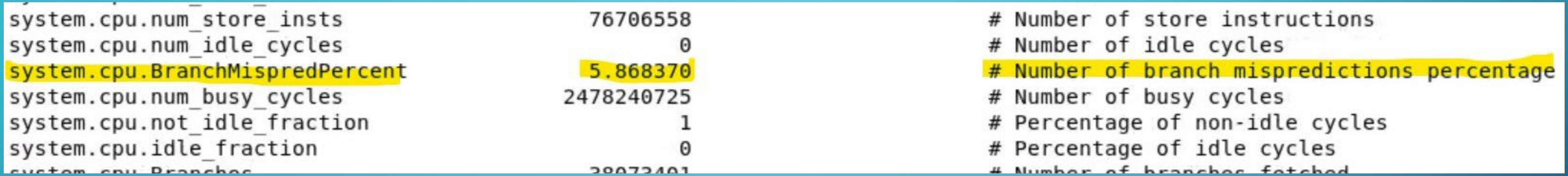
```
[system.cpu.branchPred]
type=LocalBP
BTBEntries=2048
BTBTagSize=16
RASSize=16
eventq_index=0
instShiftAmt=2
localCtrBits=2
localPredictorSize=8192
numThreads=1
```

TOURNAMENT PREDICTOR

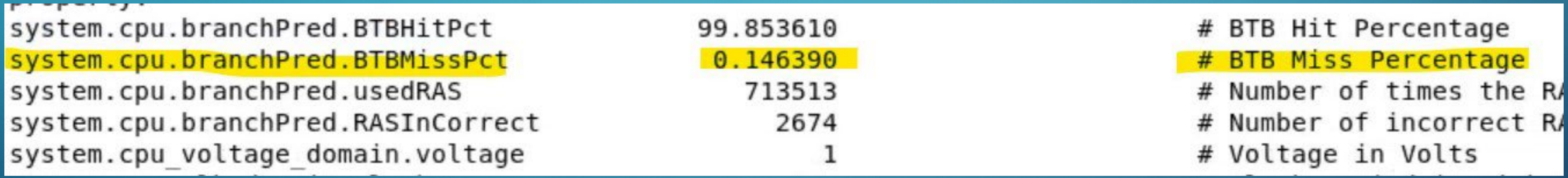


```
[system.cpu.branchPred]
type=TournamentBP
BTBEntries=4096
BTBTagSize=16
RASSize=16
choiceCtrBits=2
choicePredictorSize=1024
eventq_index=0
globalCtrBits=2
globalPredictorSize=8192
instShiftAmt=2
localCtrBits=2
localHistoryTableSize=2048
localPredictorSize=2048
numThreads=1
```

CHANGES OBSERVED IN STATS.TXT FILE AFTER ADDING BRANCH PREDICTOR SUPPORT AND MAKING CHANGES FOR THE PARAMETERS



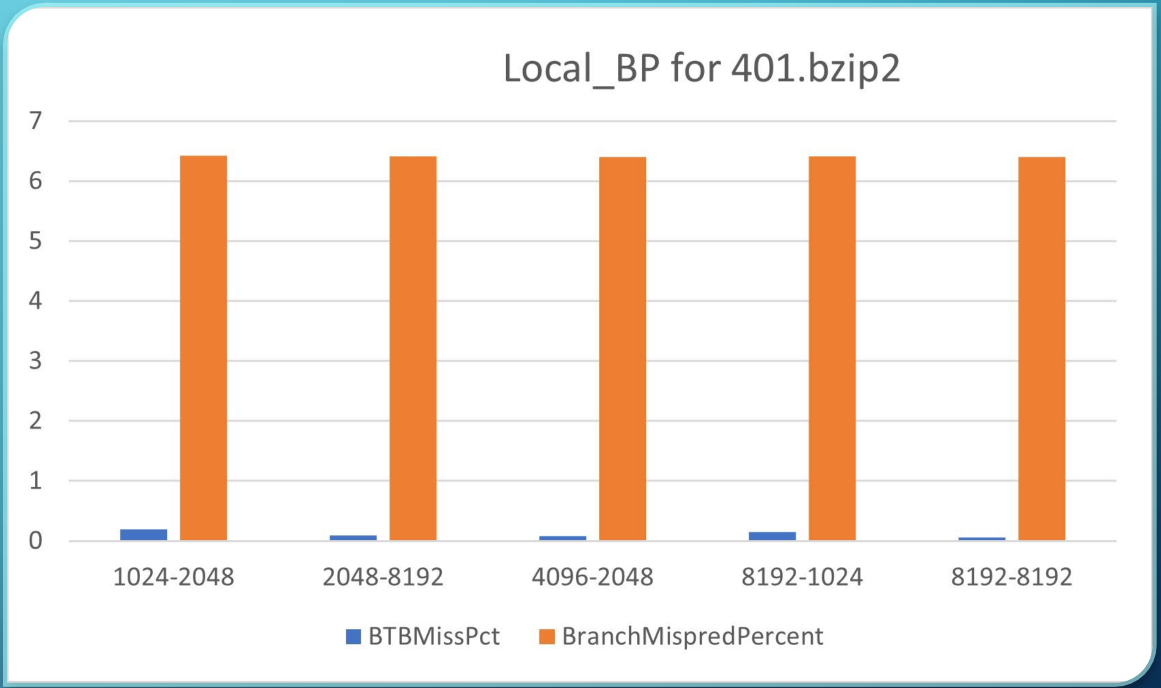
```
system.cpu.num_store_insts                  76706558                     # Number of store instructions
system.cpu.num_idle_cycles                         0                     # Number of idle cycles
system.cpu.BranchMispredPercent             5.868370                     # Number of branch mispredictions percentage
system.cpu.num_busy_cycles                2478240725                     # Number of busy cycles
system.cpu.not_idle_fraction                       1                     # Percentage of non-idle cycles
system.cpu.idle_fraction                           0                     # Percentage of idle cycles
system.cpu.Branches                         38073401                     # Number of branches fetched
```



```
system.cpu.branchPred.BTBHitPct            99.853610                     # BTB Hit Percentage
system.cpu.branchPred.BTBMissPct            0.146390                     # BTB Miss Percentage
system.cpu.branchPred.usedRAS                 713513                     # Number of times the RA
system.cpu.branchPred.RASInCorrect              2674                     # Number of incorrect RA
system.cpu_voltage_domain.voltage                  1                     # Voltage in Volts
```

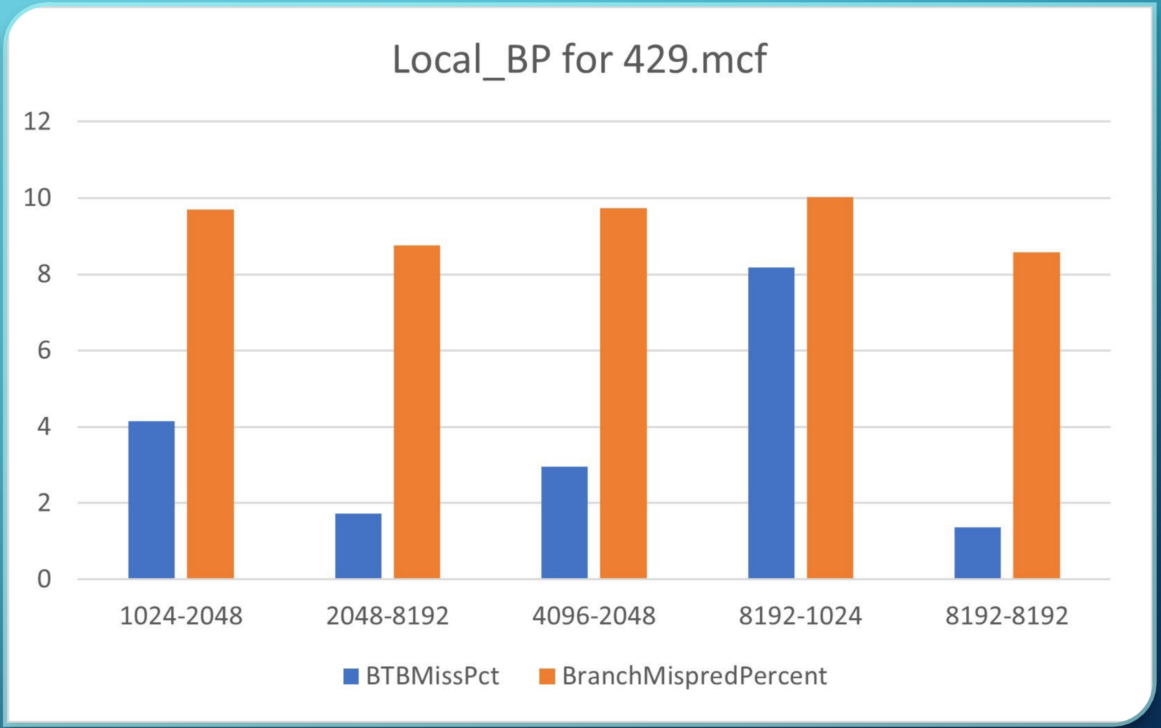
LOCAL BP FOR 401.BZIP2

- There is only very slight variation in BTB miss percentage and branch miss prediction percentage with changes in the BTB entries and local predictor size.
- However, as we increase the predictor size and the BTB buffer size the number of misses were seen to be considerably reduced.



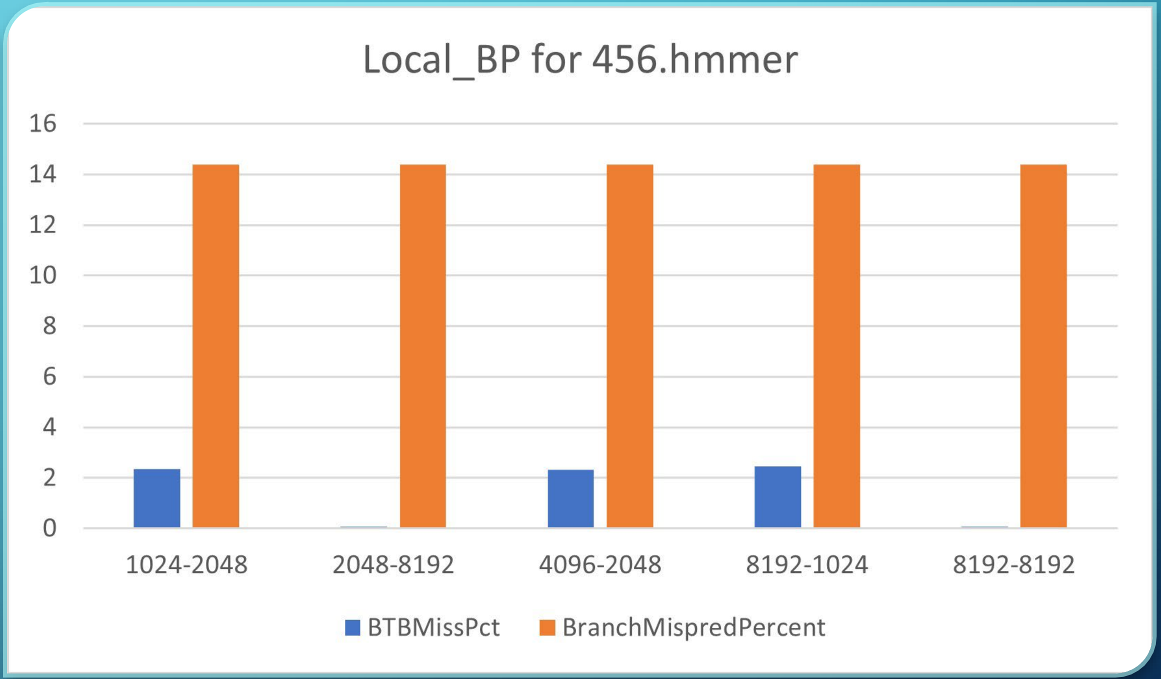
LOCAL BP FOR 429.MCF

- There is an irregular trend in BTB miss percentage and a little variation in branch miss prediction percentage with changes in the BTB entries and local predictor size.
- But as the predictor size increases from 1024 to 8192 there was a drastic drop in the number of misses.



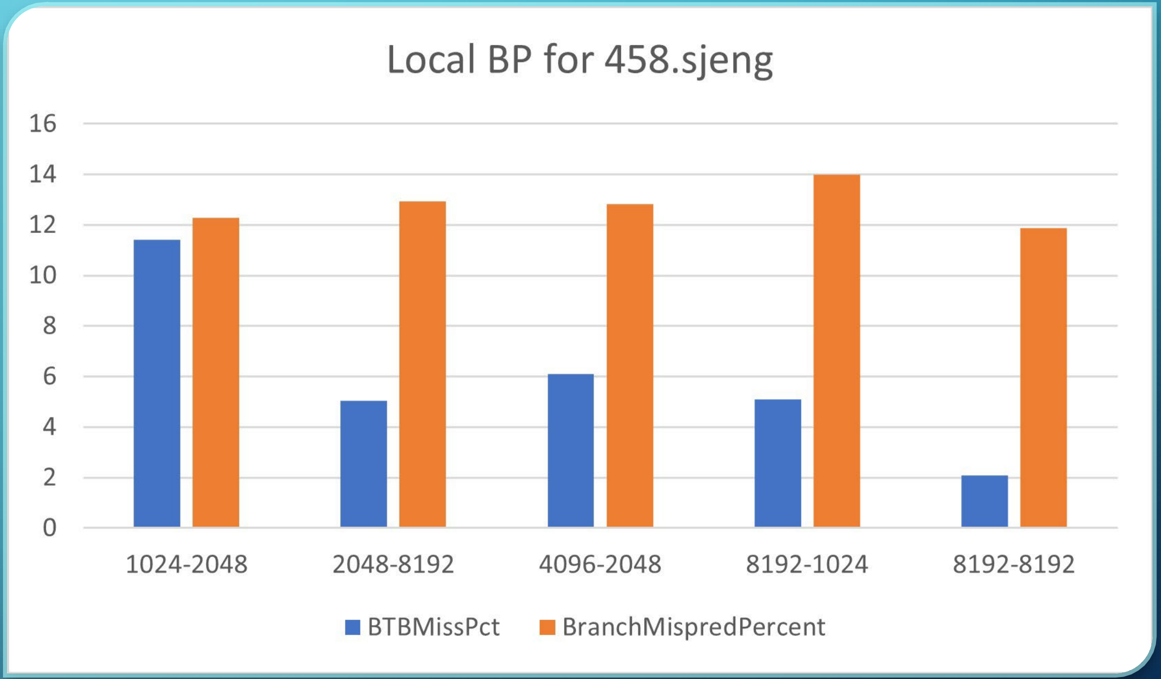
LOCAL BP FOR 456.HMMER

- For low value of local predictor size there is an increase in BTB miss percentage and branch miss prediction percentage stays almost constant with changes in the BTB entries and local predictor size.
- For this benchmark it can be observed that the between 2048-8192 and 8192-8192 configurations the variation in BTB entries has not resulted in any changes in the BTB miss percent. The bigger the predictor, the lesser the miss percentage.



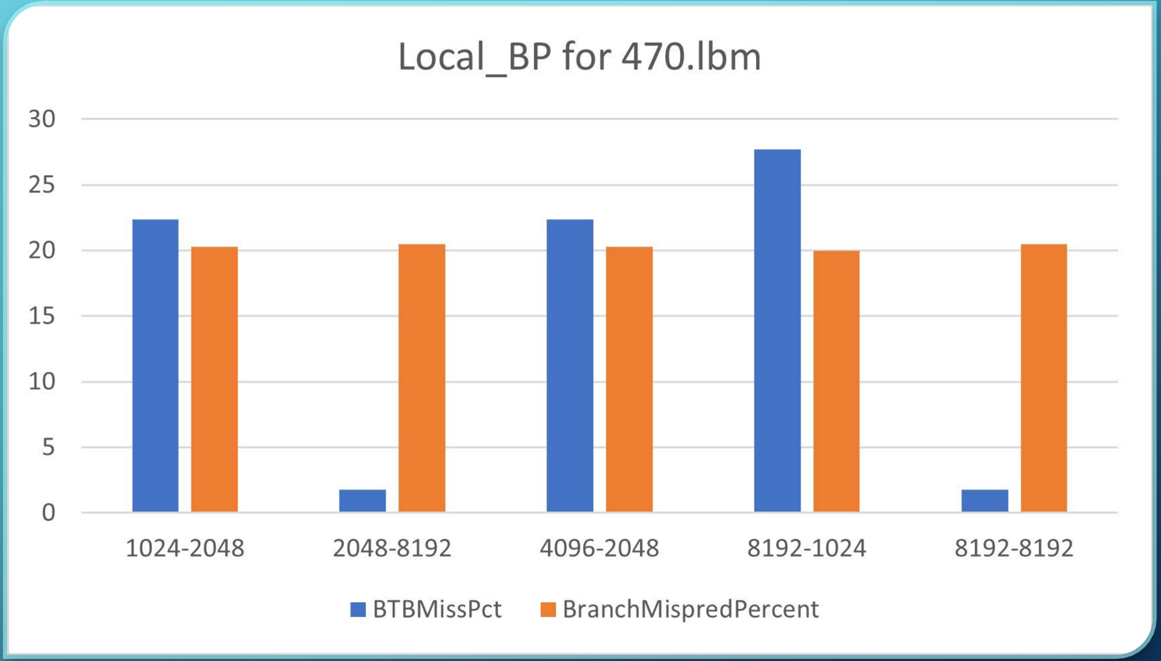
LOCAL BP FOR 458.SJENG

- There is decrease in BTB miss percentage with increase in both BTB entries and local predictor size. A very slight variation is observed in branch miss prediction percentage with changes in the BTB entries and local predictor size.
- However as the predictors size decreases, mispredictions have considerable increased as you can see in the 8192-1024 configuration.



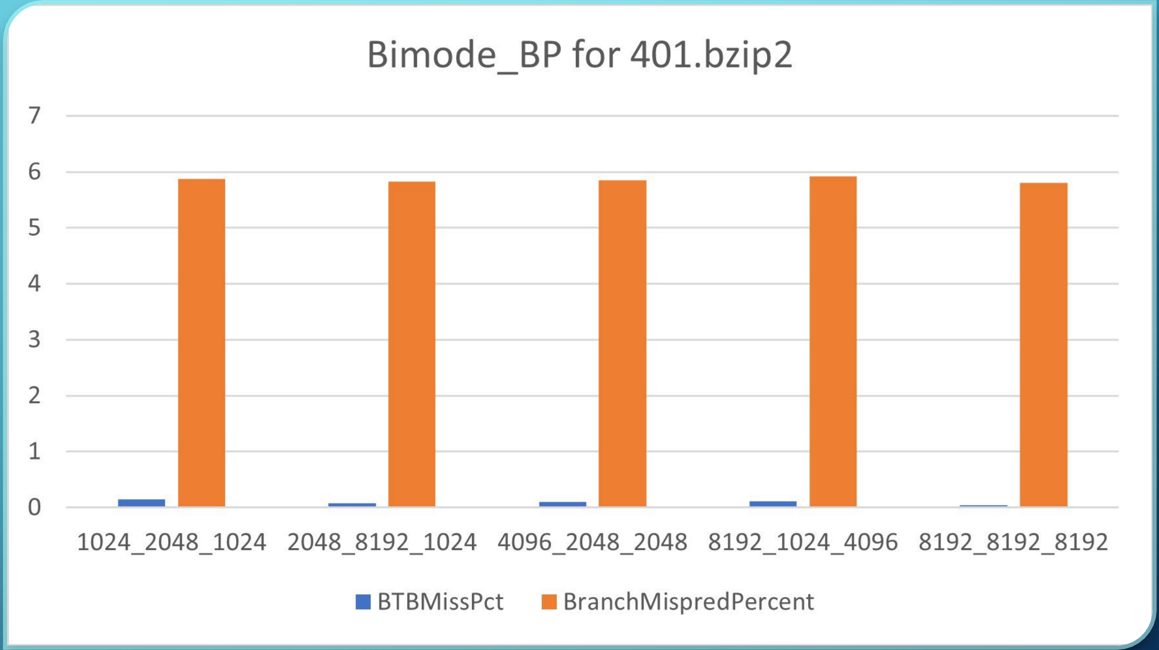
LOCAL BP FOR 470.LBM

- One of the anomalies observed for this specific benchmark is for the 8192-1024 configuration where the
- There is decrease in BTB miss percentage trend with increase local predictor size and branch miss prediction percentage stays constant with changes in the BTB entries and local predictor size.



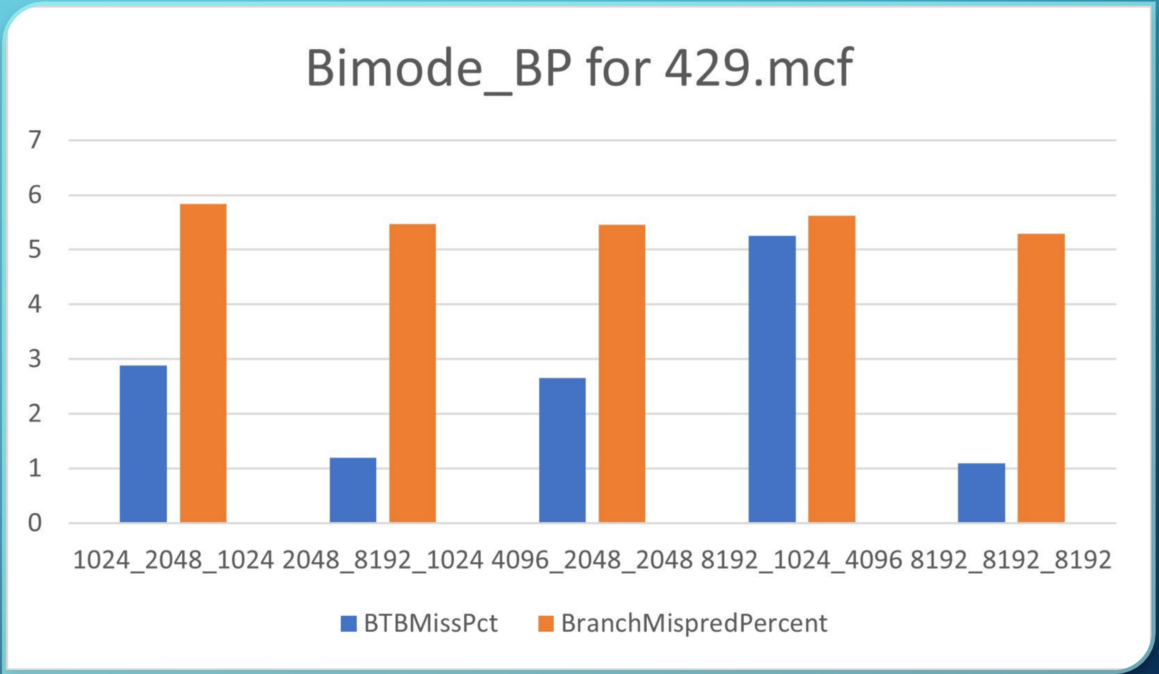
BI-MODE BP FOR 401

- There is not much variation observed in BTB miss percentage and branch miss prediction percentage with changes in the BTB entries, global predictor size and choice predictor size.
- However, a smaller global predictor size can lead to relatively more mispredictions; the same can be observed form the configuration "8192_1024_4096".



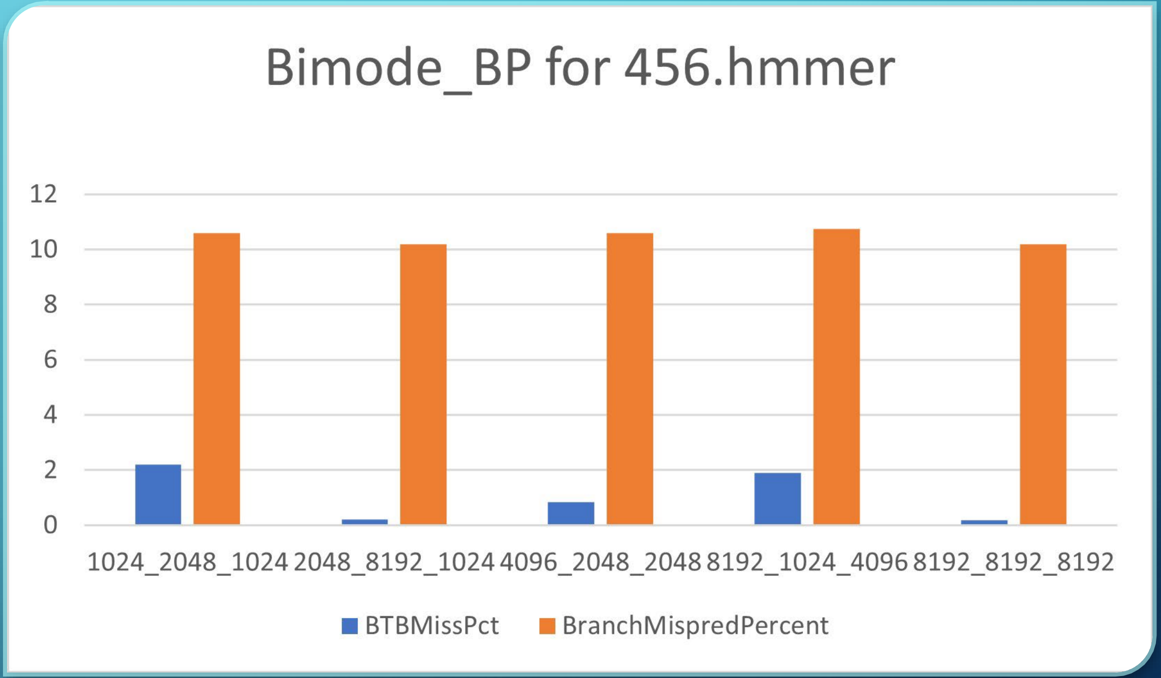
BI-MODE BP FOR 429

- There is decreasing trend in BTB miss percentage with increase in global predictor size.
- Branch miss prediction percentage doesn't change much with changes in the BTB entries, global predictor size and choice predictor size. But it seems to be the least for the for biggest sizes of the predictor.



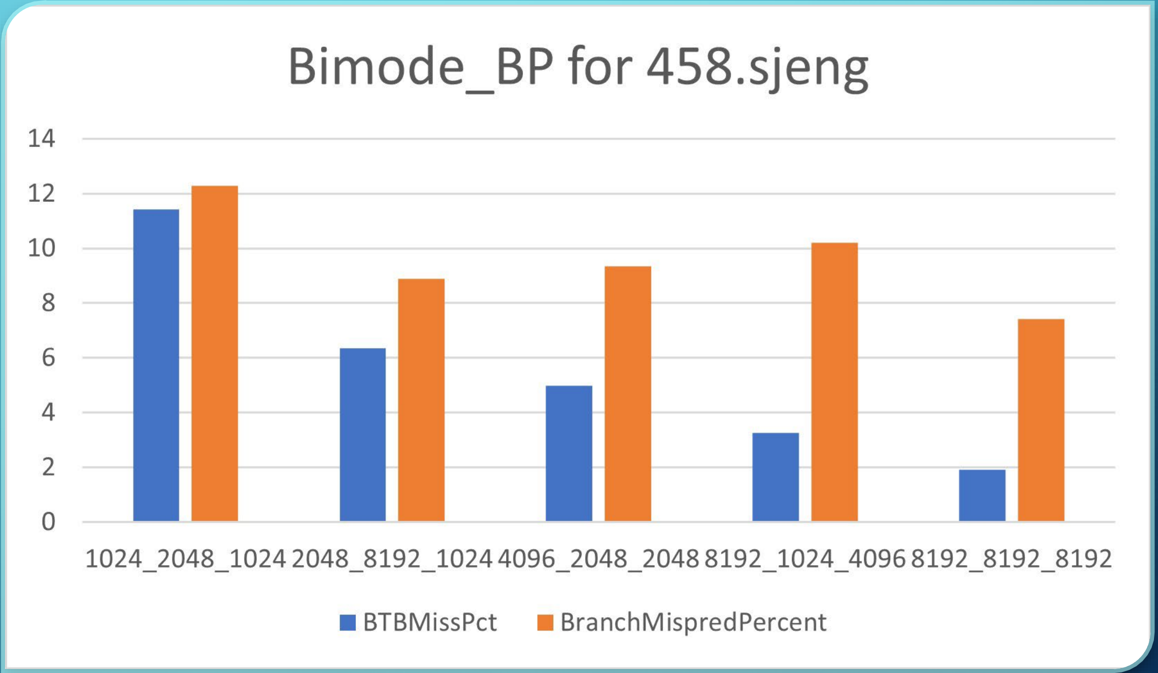
BI-MODE BP FOR 456

- With Increase in BTB entries, the BTB miss percentage seem to reduce, but that doesn’t seem to be the only factor, for bigger BTB buffer and smaller Predictor sizes the miss percentage is seen to worsen; that is increase.



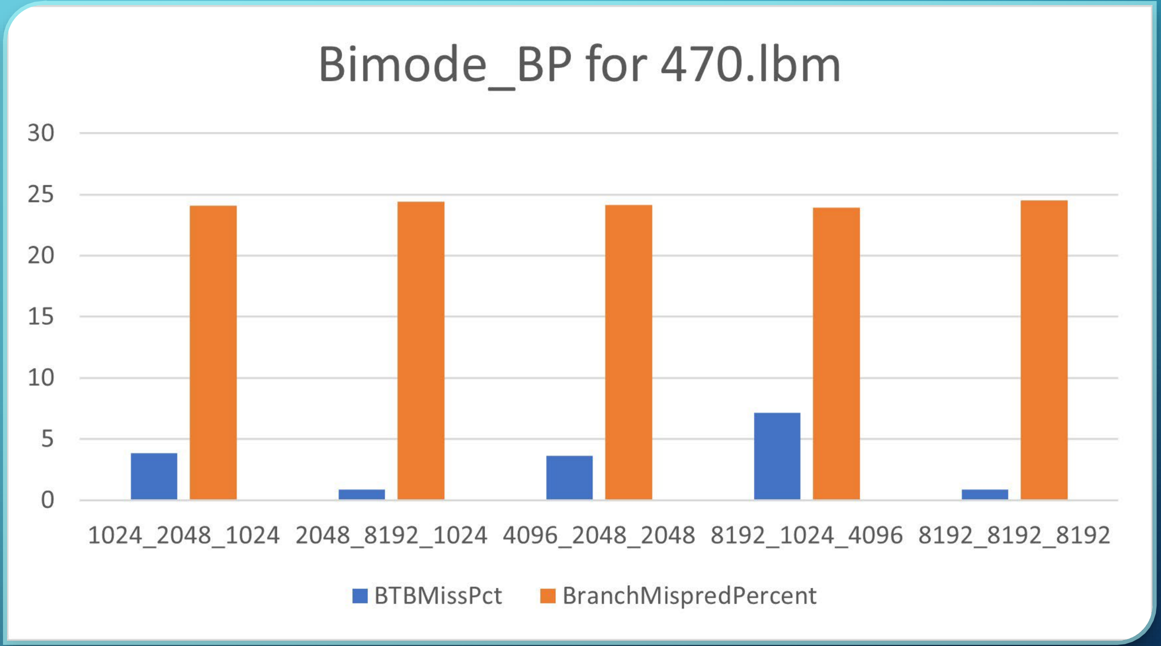
BI-MODE BP FOR 458

- There is decrease in BTB miss percentage with increase in the BTB entries, global predictor size and choice predictor size
- The branch miss prediction percentage has a slight variation with changes in the BTB entries, global predictor size and choice predictor size.



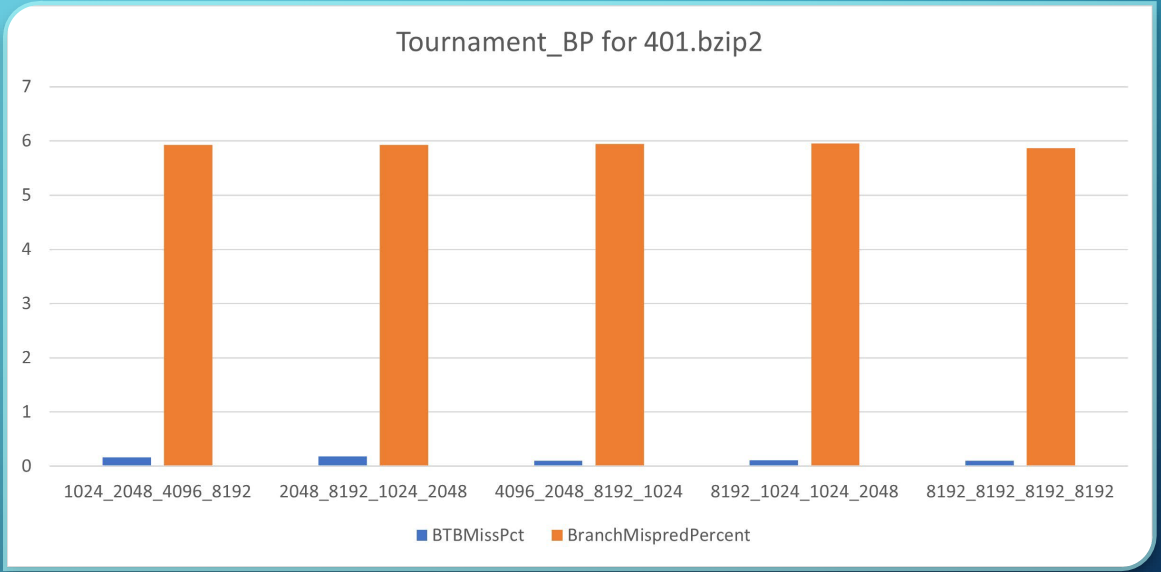
BI-MODE BP FOR 470

- There is decrease in BTB miss percentage with increase in global predictor size values and branch miss prediction percentage stays constant with changes in the BTB entries, global predictor size and choice predictor size.



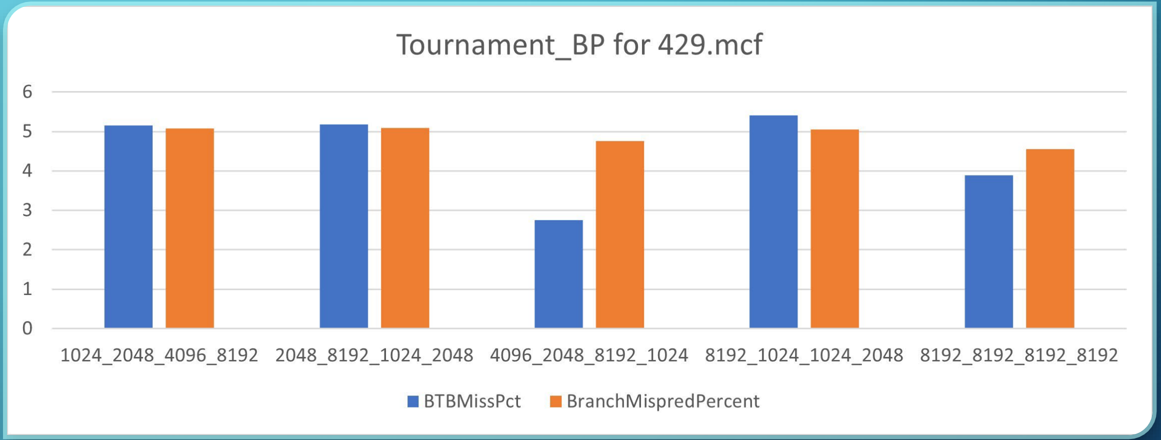
TOURNAMENT BP FOR 401

- There is not much variation in BTB miss percentage and branch miss prediction percentage with changes in the BTB entries, local predictor size, global predictor size and choice predictor size.
- However as the



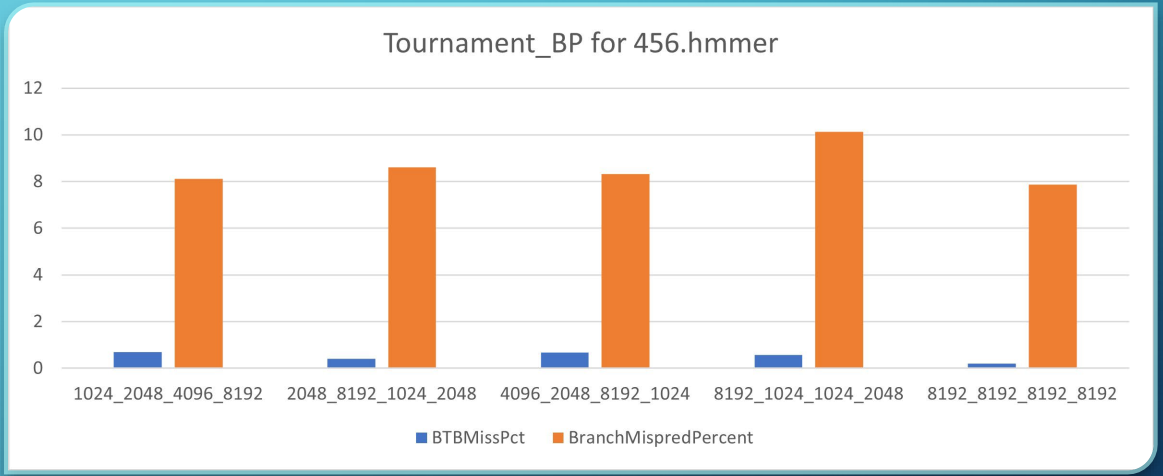
TOURNAMENT BP FOR 429

- There is irregular trend in BTB miss percentage and branch miss prediction percentage is constant with changes in the BTB entries, local predictor size, global predictor size and choice predictor size.
- Between configurations “4096_2048_8192_1024” and “8192_8192_8192_8192” its is seen that with a smaller BTB buffer and smaller local predictor we were still able to achieve a better miss percent and almost the same misprediction rate.



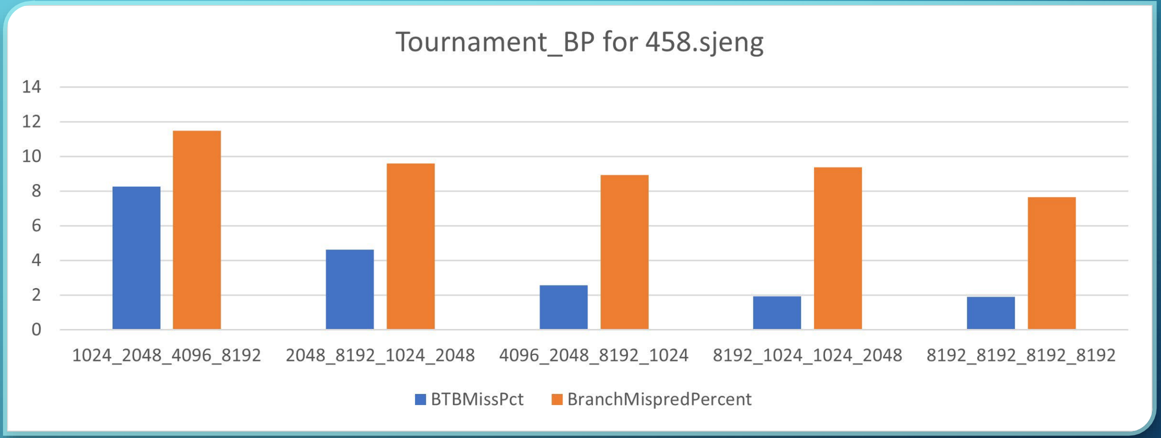
TOURNAMENT BP FOR 456

- There is slight variation in BTB miss percentage and branch miss prediction percentage with changes in the BTB entries, local predictor size, global predictor size and choice predictor size.
- Best values have been observed when all the considered parameters are at their maximum. See configuration "8192_8192_8192_8192".



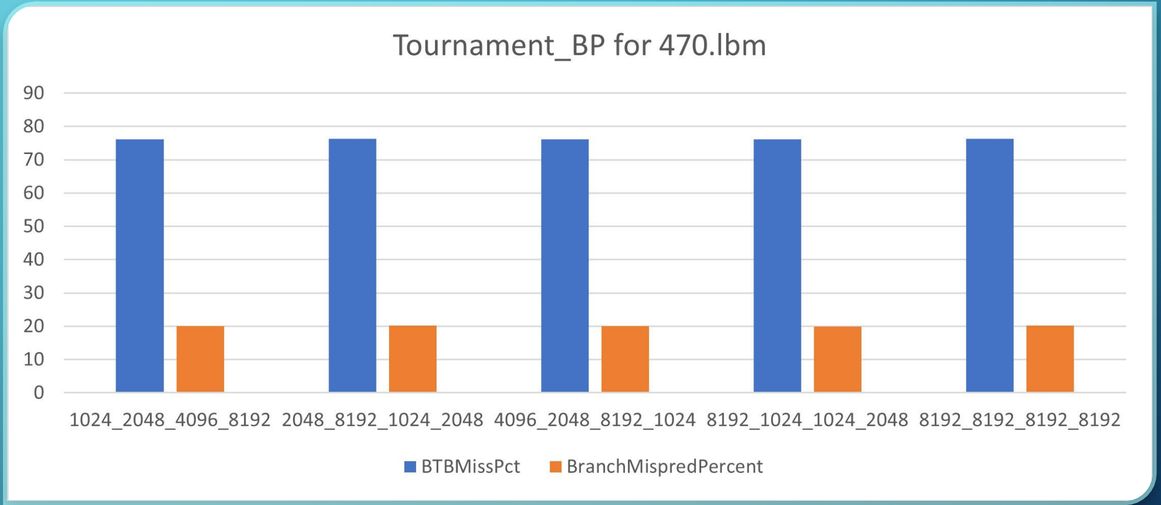
TOURNAMENT BP FOR 458

- There is decreasing trend on in BTB miss percentage and branch miss prediction percentage with increasing trend in the BTB entries, local predictor size, global predictor size and choice predictor size.



TOURNAMENT BP FOR 470

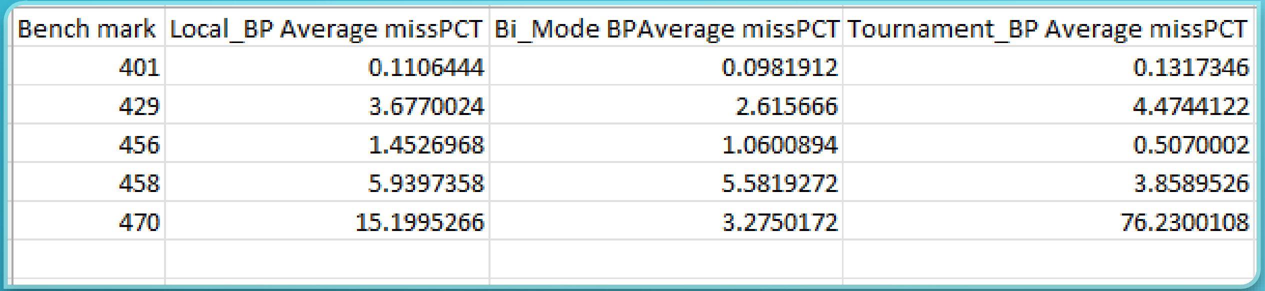
- There is no variation in BTB miss percentage and branch miss prediction percentage with changes in the BTB entries, local predictor size, global predictor size and choice predictor size.



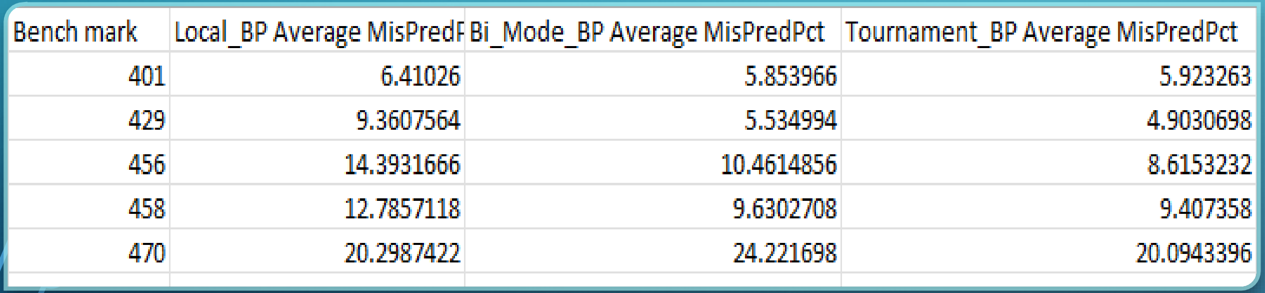
ADDITIONAL OBSERVATIONS

- For benchmark 401.bzip2, the changes in the BTB miss percentage is negligible and Branch miss prediction percentage is high for all Local BP, Tournament BP and BiMode BP.
- For 470.lbm, BTB miss percentage is very high for Local BP and Tournament BP but for BiMode BP it is very little.

CONCLUSION PART-1



Bench mark	Local_BP Average missPCT	Bi_Mode BPAverage missPCT	Tournament_BP Average missPCT
401	0.1106444	0.0981912	0.1317346
429	3.6770024	2.615666	4.4744122
456	1.4526968	1.0600894	0.5070002
458	5.9397358	5.5819272	3.8589526
470	15.1995266	3.2750172	76.2300108



Bench mark	Local_BP Average MisPredF	Bi_Mode_BP Average MisPredPct	Tournament_BP Average MisPredPct
401	6.41026	5.853966	5.923263
429	9.3607564	5.534994	4.9030698
456	14.3931666	10.4614856	8.6153232
458	12.7857118	9.6302708	9.407358
470	20.2987422	24.221698	20.0943396

- Finally, we have used all the accumulated data to derive the average BTBmissPCT and BranchMisPred values of each of the predictor on all the five benchmarks.
- As it can be inferred from data, Tournament Branch Predictor is the most efficient Branch Predictor when tested over all the benchmarks.

CONCLUSION PART-2

- The performance of a branch predictor depends on several factors such as the characteristics of the program being executed, the memory hierarchy of the system, and the design of the predictor itself.
- Tournament BP has a combination of Local, Global and Choice predictor sizes which makes it more accurate than Local BP and Bi-Mode BP.
- Tournament BP can be easily scaled to handle large branch history tables, making it well-suited for modern processors with deep pipelines and large instruction windows.
- Tournament BP is adaptable, which means it adjusts itself according to the program.
- Overall, Tournament BP has better edge over Local BP and BiMode BP because it has higher accuracy, is more adaptable and can be implemented with acceptable complexity on modern processors.